Noname manuscript No.
(will be inserted by the editor)

P Colonies and kernel P Systems

Erzsébet Csuhaj-Varjú · Marian Gheorghe · Raluca Lefticaru

Received: date / Accepted: date

Abstract P colonies, tissue-like P systems with very simple components, have received constant attention from the membrane computing community and in the last years several new variants of the model have been considered. Another P system model, namely kernel P system, integrating the most successfully used features of membrane systems,has recently attracted interest and some important developments have been reported. In this paper we study connections among several classes of P colonies and kernel P systems, by showing how the behaviour of these P colony systems can be represented as kernel P systems. An example illustrates the way it is modelled by using P colonies and kernel P systems and some properties of it are formally proved in the later approach.

1 Introduction

Membrane systems were introduced in [22] as a new natural computing paradigm inspired by the compartmentalised structure of the living cells and the main bio-chemical interactions occurring within and across compartments. A monograph [23] presenting the main developments at the level of early 2000 was

E. Csuhaj-Varjú
Department of Algorithms and Their Applications, Faculty of Informatics, ELTE Eötvös Loránd University, Budapest 1117, Hungary
E-mail: csuhaj@inf.elte.hu

M. Gheorghe
School of Electrical Engineering and Computer Science, University of Bradford, Bradford BD7 1DP, United Kingdom
E-mail: m.gheorghe@bradford.ac.uk

R. Lefticaru
School of Electrical Engineering and Computer Science, University of Bradford, Bradford BD7 1DP, United Kingdom
E-mail: r.lefticaru@bradford.ac.uk

published. A comprehensive description of the key research areas covering both theoretical aspects as well as applications appears in this handbook [26]. More specific developments may be found in research papers, referring to generalised forms of rewriting P systems [20], rewriting tissue P systems [21], control languages used for P systems [18].

A *P colony* represents a membrane system model with communities of cells communicating with a shared environment by means of simple rules and using a limited number of symbols in each cell [14,5]. An overview of the main developments of the model is presented in [15], a recent survey of the area can be found in [3]. Two more P colony models are of interest in the context of this paper, namely *P colonies with senders and consumers*, and *P colonies with evolving environment*. The first model [4], considers specific communication rules acting in one single direction, either from the cell to the environment (sending cell) or vice-versa (consuming cell). The later model considers the case of an environment whereby its contents might be affected not only by the exchanges with cells, but also by some evolving rules acting similarly to rules of context-free Lindenmayer systems (L systems). The behaviour and the computational power of this model are investigated in [2].

Kernel P systems (kP systems, for short) [8,9] aim to integrate in a coherent and elegant manner some of the most successfully used features of P systems.

The kP system model is also supported by a modelling language, called kP-Lingua, capable of mapping a kP system model specification into a machine readable representation. Furthermore, kP systems are supported by a software framework, kPWorkbench [13], which integrates a set of related simulation and verification methods and tools.

After being introduced [8,9], kP systems have been investigated from various research angles. Their relationships with other classes of membrane systems have been investigated - firstly, membrane systems with active membranes and neural-like membrane systems have been mapped into kP systems [9,10], then generalised communicating P systems have been connected with kP systems [16]. Tools, such as kPWorkbench [13], have linked modelling aspects with formal verification and model checking [6,13,7]. Various applications have been considered, such as 3-colouring problem [11], sorting algorithms [9,7], simple broadcasting [13], or synthetic biology paradigms - genetic logic gates [12].

In this paper we continue the investigation of the P colony and kP system models by considering the relationship between them. We show how the behaviour of P colonies of arbitrary capacity, P colonies with senders and consumers, and P colonies with capacity 2 and evolving environment is mapped into corresponding equivalent kP systems. Finally, we consider the problem of modelling the synchronisation aspects occurring in a producer/consumer problem by using adequate P colony and kP systems, exhibiting equivalent behaviour. Formal properties are investigated using the model checking facilities provided by the kPWorkbench tool associated with kP systems.

The paper is organised as follows. In Section 2 we introduce the definitions of a kernel P system and a P colony. In Section 3 we study the relationships among the above mentioned classes of P colonies and kP systems. Section 4 introduces the producer/consumer problem and investigates its representation using P colony with senders and consumers and kP systems, as well as some formal properties. Finally, Section 5 concludes the paper.

2 Kernel P Systems and P Colonies - Main Concepts and Definitions

Standard P system concepts and standard notions such as strings, multisets, rewriting rules, and computation are well-known and we refer to [23] for their formal notations and precise definitions. The kP system concepts and definitions used in this paper are from [8,9]. Some of them are slightly changed and this will be mentioned when these concepts are discussed.

2.1 kP System Basic Definitions

We start by introducing the concept of a *compartment type* utilised later in defining the compartments of a kP system.

Definition 1 T is a *set of compartment types*, $T = \{t_1, \ldots, t_s\}$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, and an execution strategy, σ_i, defined over $Lab(R_i)$, the labels of the rules of R_i.

The compartments that appear in the definition of the kP systems will be constructed using compartment types introduced by Definition 1. Each compartment, C, will be defined by a tuple (t, w), where $t \in T$ is the type of the compartment and w the initial multiset of it. The types of rules and the execution strategies occurring in the compartment types will be introduced and discussed later.

Definition 2 A *kP system* of degree n is a tuple

$$k\Pi = (A, \mu, C_1, \ldots, C_n, i_0),$$

where A is a finite set of elements called *objects*; μ defines the *initial membrane structure*, which is a graph, (V, E), where V are vertices indicating compartments of the kP system, and E edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a *compartment* of the kP system, as presented above; i_o is the label of the *output compartment*, where the result is obtained.

2.2 kP System Rules

Each rule occurring in a kP system definition has the form $r\ \{g\}$, where r stands for the rule itself and g is its **guard**. The guards are constructed using multisets over A, as operands, and relational or Boolean operators. The definition of the guards is now introduced. We start with some notations.

For a multiset w over A and an element $a \in A$, we denote by $|w|_a$ the number of objects a occurring in w. Let us denote $Rel = \{<, \leq, =, \neq, \geq, >\}$, the set of relational operators, $\gamma \in Rel$, a relational operator, and a^n a multiset. We first introduce an *abstract relational expression.*

Definition 3 If g is the *abstract relational expression* denoting γa^n and w a multiset, then the guard g applied to w denotes the *relational expression* $|w|_a \gamma n$.

The abstract relational expression g is true for the multiset w, if $|w|_a \gamma n$ is true.

We consider now the following Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction). An *abstract Boolean expression* is defined by one of the following conditions

- any abstract relational expression is an abstract Boolean expression;
- if g and h are abstract Boolean expressions then $\neg g$, $g \wedge h$ and $g \vee h$ are abstract Boolean expressions.

The concept of a guard, introduced here, is a generalisation of the promotor and inhibitor concepts utilised by some variants of P systems.

Definition 4 If g is an *abstract Boolean expression* containing g_i, $1 \leq i \leq q$, abstract relational expressions and w a multiset, then g applied to w means the *Boolean expression* obtained from g by applying g_i to w for any $i, 1 \leq i \leq q$.

As in the case of an abstract relational expression, the guard g is true with respect to the multiset w, if the abstract Boolean expression g applied to w is true.

Example 1 If g is the guard defined by the abstract Boolean expression $\geq a^5 \wedge < b^3 \vee \neg > c$ and w a multiset, then g applied to w is true if it has at least 5 a's and no more than 2 b's or no more than one c.

Definition 5 A rule from a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$ can have one of the following types:

- (a) **rewriting and communication** rule: $x \to y\ \{g\}$, where $x \in A^+$ and y has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, $h \geq 0$, $a_j \in A$ and t_j, $1 \leq j \leq h$, indicates a compartment type from T (see Definition 2) of compartments linked to the current one; t_j might also indicate the type of the current compartment, $C_{t_{l_i}}$; if a link does not exist (i.e., there is no link between the two compartments in E) then the rule is not applied; if a target, t_j, refers to a compartment type that appears in more than one compartments connected to C_{l_i}, then one of them will be non-deterministically chosen;

- (b) **structure changing rules**; the following types of rules are considered:
  - (b1) **membrane division** rule: $[x]_{t_{l_i}} \rightarrow [y_1]_{t_{i_1}} \dots [y_p]_{t_{i_p}} \{g\}$, where $x \in A^+$ and $y_j \in A^*$, $1 \leq j \leq p$; the compartment C_{l_i} will be replaced by p compartments; the j-th compartment, $1 \leq j \leq p$, of type t_{i_j} contains the same objects as C_{l_i}, but x, which will be replaced by y_j; all the links of C_{l_i} are inherited by each of the newly created compartments;
  - (b2) **membrane dissolution** rule: $[]_{t_{l_i}} \rightarrow \lambda \{g\}$; the compartment C_{l_i} will be destroyed together with its links;
  - (b3) **link creation** rule: $[x]_{t_{l_i}}; []_{t_{l_j}} \rightarrow [y]_{t_{l_i}} - []_{t_{l_j}} \{g\}$; the current compartment is linked to a compartment of type t_{l_j} and x is transformed into y; if more than one compartment of type t_{l_j} exist and they are not linked with $C_{t_{l_i}}$, then one of them will be non-deterministically picked up; g is a guard that refers to the compartment of type t_{l_i};
  - (b4) **link destruction** rule: $[x]_{t_{l_i}} - []_{t_{l_j}} \rightarrow [y]_{t_{l_i}}; []_{t_{l_j}} \{g\}$; is the opposite of link creation and means that the compartments are disconnected.

When in a rewriting and communication rule one of the right hand side elements (a_j, t_j), $1 \leq j \leq h$, is such that $t_j = t_{l_i}$ then this is simply written as a_j.

The membrane division is defined slightly differently here compared to [8, 9], where each y_j, $1 \leq j \leq p$, is composed of objects with target compartments.

In this paper we will be using only rewriting and communication rules (a).

2.3 kP System Execution Strategies

In kP systems the way in which rules are executed is defined for each compartment type t from T – see Definition 1. As in Definition 1, $Lab(R)$ is the set of labels of the rules R.

Definition 6 For a compartment type $t = (R, \sigma)$ from T and $r \in Lab(R)$, $r_1, \dots, r_s \in Lab(R)$, the *execution strategy*, σ, is defined by the following

- $\sigma = \lambda$, means no rule from the current compartment will be executed;
- $\sigma = \{r\}$ – the rule r is executed;
- $\sigma = \{r_1, \dots, r_s\}$ – one of the rules labelled $r_1, \dots, r_s$ will be non-deterministically chosen and executed; if none is applicable then nothing is executed; this is called *alternative* or *choice*;
- $\sigma = \{r_1, \dots, r_s\}^*$ – the rules are applied an arbitrary number of times (*arbitrary parallelism*);
- $\sigma = \{r_1, \dots, r_s\}^\top$ – the rules are executed according to the *maximal parallelism* strategy;
- $\sigma = \sigma_1 \& \dots \& \sigma_s$, means executing sequentially $\sigma_1, \dots, \sigma_s$, where σ_i, $1 \leq i \leq s$, describes any of the above cases; if one of σ_i fails to be executed then the rest is no longer executed;
- for any of the above σ strategy only one single structure changing rule is allowed.

Remark 1 For a kP system of degree n, as in Definition 2, an $n-$tuple $(x_1, \ldots, x_n)$, where $x_i \in A$, $1 \leq i \leq n$, are multisets over A, is called a *configuration* of $k\Pi$.

Remark 2 A computation, as usual in membrane computing, is defined as a sequence of finite steps starting from the initial configuration, with the initial multisets distributed in compartments – Definition 1 and Definition 2. In each step the rules are selected according to the execution strategy in each compartment. The result of a computation will be the number of objects collected in the output compartment. For a kP system, $k\Pi$, the set of all these numbers will be denoted by $N(k\Pi)$.

P colony [5] represents a simple membrane system model with communities of cells communicating with a shared environment. Here we consider a restricted version of it, called *P colony without checking rules*. More on this model can be found in [15]. The other P colony models will be also simplified versions of the original models, however all of them are computationally complete. We use these simplified models as we are interested in establishing connections with kernel P systems and prefer to use the simplest P colony models that are computationally complete.

Definition 7 A *P colony* (without checking rules) is an $n + 3$-tuple, $n \geq 1$

$$\Pi = (O, e, F, B_1, \ldots, B_n),$$

where

- O is a finite set, called the alphabet of *objects*;
- $e \in O$ is the *environment object*;
- $F \subseteq O$ is the set of *final objects*;
- B_i, $1 \leq i \leq n$, is called a *cell* of Π and $B_i = (o_i, P_i)$, where o_i is a multiset over $\{e\}$, called the *initial multiset* of the cell and P_i is a finite set of *programs*, $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$. Each program, $p_{i,j}$, $1 \leq j \leq k_i$, consists of a finite multiset of rules of the following forms:
  - (a) $a \to b$, $a, b \in O$, is called *internal point mutation* or *evolution*, specifying that an object a inside the cell B_i is changed to b;
  - (b) $c \leftrightarrow d$, $c, d \in O$, is called *one object exchange with the environment* or *communication*, specifying that if c is contained inside the cell B_i and d is present in the environment, then c is sent out of the cell into the environment while d is brought inside the cell from the environment.

The multisets o_i, $1 \leq i \leq n$, have the same cardinality, called *capacity* of Π. The number of rules in each program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, coincides with the capacity of Π.

An $n + 1$-tuple $(x_E, x_1, \ldots, x_n)$, where $x_E \in (O \setminus \{e\})^*$, $x_i \in O^*$ are finite multisets, is called a *configuration* of Π. At the *initial configuration*, the environment contains arbitrarily many copies of e and each cell contains inside as many objects e as the capacity of Π.

The P colony works with direct changes of its configurations, called *transitions*. To obtain a new configuration by a transition, the programs of the cells are used in the non-deterministic maximally parallel manner, i.e., each cell which is able to use one of its programs should use one. The use of a program means the parallel application of the rule(s) of the program to the object(s) inside the cell. A sequence of transitions starting from the initial configuration is a *computation*. A computation is successful if it is *halting*, i.e., if a configuration is obtained where no cell can use any program. The *result* of a successful computation is the number of copies of objects from F present in the halting configuration. The set of numbers obtained as results of successful computations of a P colony Π is denoted by $N(\Pi)$.

We introduce now the concept of *P colonies with senders and consumers*. The definition appears for the first time in [4].

Definition 8 A P colony with senders and consumers, Π, is a P colony, as in Definition 7, with the following constraints:

- each cell B_i, $1 \leq i \leq n$, contains always 2 objects (the capacity of Π is 2);
- each program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, from cell B_i is one of the following two types:
  - *deletion* or *consumer* program $< a_{in}; bc \rightarrow d >$, with $a, b, c, d \in O$ and the meaning that object a from the environment is consumed, i.e., it is brought into cell B_i and the objects b, c are transformed into d inside of B_i;
  - *insertion* or *sender* program $< a_{out}; b \rightarrow cd >$, with $a, b, c, d \in O$ and the meaning that object a is sent out of B_i into the environment and b is transformed into objects c and d inside of B_i.
- every program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, of cell B_i is of the same type. A cell with only insertion programs is called a *sender* and a cell with only deletion programs is called a *consumer*.

In this model the Turing completeness is obtained for P colonies whereby each cell has only either consumer or producer programs. So, the cells are either consumers or senders.

Next we provide the notion of a *P colony with evolving environment* – for details regarding this computational model we refer to [2]. In the basic model the environment does not change under functioning, it can be altered only by the activity of the cells. However, to extend the concept of P colonies with a dynamically evolving environment is a natural idea. In this case the environment has its own rules for evolution which do not depend on the cells.

These rules are given as multiset rewriting rules, usually given as a multiset 0L scheme. A multiset 0L scheme is a pair (V, P), where V is the alphabet of 0L scheme and P is a complete set of context-free multiset rewriting rules over V, i.e., where P is a finite set of multiset rewriting rules over V of the form $a \rightarrow w$, where $a \in V$ and $w \in V^*$ and for each $a \in V$ there exists at least one rule in P. For $w_1, w_2 \in V^*$, we write $w_1 \Rightarrow w_2$ if $w_1 = a_1 a_2 \dots a_n$ and $w_2 = \alpha_1 \alpha_2 \dots \alpha_n$, for $a_i \rightarrow \alpha_i \in P, 1 \leq i \leq n$, i.e., w_1 derives w_2 in one step (directly).

Before providing the definition, we note that in the literature P colonies with evolving environment are also called generalized P colonies. In this paper, to emphasize the difference between standard P colonies and these developed versions, we will use the term P colony with evolving environment and we consider the case when the capacity of the P colony is 2..

Definition 9 A P colony $\Pi = (O, e, F, o_E, D_E, B_1, \ldots, B_n)$ with evolving environment is a combination of the concept of a P colony in Definition 7 and the concepts of a P colony with senders and consumers in Definition 8 with the following additional constraints:

- $o_E \in (O \setminus \{e\})^*$,
- $D_E = (O \setminus \{e\}, P_E)$ is a multiset $0L$ scheme,
- each cell B_i, $1 \leq i \leq n$, contains always 2 objects (the capacity of Π is 2); each program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$ of B_i has a finite set of programs of exactly one of the following three types:
  - *deletion* or *consumer* program $< a_{in}; bc \to d >$, with $a, b, c, d \in O$ and the meaning that object a from the environment is consumed, i.e., it is brought into cell B_i and the objects b, c are transformed into d inside of B_i;
  - *insertion* or *sender* program $< a_{out}; b \to cd >$, with $a, b, c, d \in O$ and the meaning that object a is sent out of B_i into the environment and b is transformed into objects c and d inside of B_i.
  - a program with two rules which are evolution rules (of the form $< a \to b >$) and/or communication rules (of the form $< a \leftrightarrow b >$), as given in Definition 7.

The *initial configuration* of a P colony with evolving environment is the $(n+1)$-tuple $(o_E, o_1, \ldots, o_n)$, o_i, $1 \leq i \leq n$, as given in Definition 7.

By applying programs, the P colony with evolving environment passes from one configuration to some other configuration. Those objects in the environment which are not affected by any program in the given step are rewritten by the multiset 0L scheme D_E. As in the previous cases, the systems works with the non-deterministic maximally parallel manner, i.e., at each step a maximal number of cells performs one of its programs in parallel (the program is chosen non-deterministically out of the applicable ones). Those objects in the environment which are different from the basic object, e, and are not affected by any program, are changed according to the rules of the multiset $0L$ scheme.

A computation is a sequence of consecutive configurations starting from the initial configuration and ending in a configuration when no cell has any applicable program. Notice that in this case the work of the P colony itself does not necessarily stop, since the multiset $0L$ scheme may continue changing the non-environment objects in the environment, if such an object exists.

The result of the computation is the number of copies of final objects in the environment in such a configuration when no cell is able to perform at least one program.

3 Relationships between various P colonies and kernel P systems

We start by looking at the relationships between P colonies of capacity h, $h \geq 1$, and kP systems. It may be observed from the proofs of these results that we use kP systems with only rewriting and communication rules. Guards are only used in the last theorem.

Theorem 1 *For every P colony $\Pi = (O, e, F, B_1, \ldots, B_n)$, $n \geq 1$, of capacity h, $h \geq 1$, one can construct a kP system $k\Pi = (A, \mu, C_1, C_2, 2)$, such that $N(\Pi) = N(k\Pi)$ holds.*

Proof Let us consider $\Pi = (O, e, F, B_1, \ldots, B_n)$, $n \geq 1$, a P colony of size n and capacity h. It is known that the result of a computation in Π is obtained in the environment, consisting only of objects from F. The cell B_i, $1 \leq i \leq n$, has the initial multiset o_i consisting of h objects and each program $p_{i,j}$, $1 \leq j \leq k_i$, from the set of programs, P_i, consists of h rules.

We will build a kP system with two compartments

$$k\Pi = (A, \mu, C_1, C_2, 2)$$

where C_1 is simulating the behaviour of the P colony, Π, and C_2 is collecting the result that coincides with the result of the P colony, Π.

The objects of the kP system are $A = F \cup \{f' | f \in F\} \cup \{[a, i] | a \in O, 0 \leq i \leq n\}$. For each $a \in O$, when it belongs to cell B_i, $1 \leq i \leq n$, or the environment, 0, it will be denoted by $[a, i]$ or $[a, 0]$, respectively.

The membrane structure, μ, provides a connection between the two compartments, C_1 and C_2, allowing them to exchange objects, where $C_i = (t_i, w_i)$, $i = 1, 2$. Each t_i, $1 \leq i \leq 2$, the type of compartment C_i, is given by $t_i = (R_i, \sigma_i)$, where R_i is the set of rules and σ_i the execution strategy. These will be introduced later on. The second component of C_i is the initial multiset of the compartment. We have $w_1 = [o_1, 1] \ldots [o_n, n]$ and $w_2 = \lambda$.

The type, t_1, of the compartment C_1 has the set of rules, R_1, given below.

Next we define the rules of $k\Pi$. This will be done in several steps. Notice first that any program $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, consists of h rules $r_{i,j,k}$, $1 \leq k \leq h$ that can be evolution rules or communication rules. Furthermore, if a program is applied, then all of its rules should be performed in parallel.

First we will define rules $r'_{i,j,k}$ to rules $r_{i,j,k}$ in program $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, $l \leq k \leq h$. When

1. $r_{i,j,k} : a \to b$ with $a, b \in O$, then let $r'_{i,j,k} : [a, i] \to [b, i]$.
2. $r_{i,j,k} : c \leftrightarrow d$ with $c, d \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [c, i][d, 0] \to [c, 0][d, i]$.
3. $r_{i,j,k} : c \leftrightarrow e$ with $c \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [c, i] \to [c, 0][e, i]$.
4. $r_{i,j,k} : e \leftrightarrow d$ with $d \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [e, i][d, 0] \to [d, i]$.
5. $r_{i,j,k} : e \leftrightarrow e$ then let $r'_{i,j,k} : [e, i] \to [e, i]$.
6. $r_{i,j,k} : f \leftrightarrow d$ with $d \in (O \setminus (F \cup \{e\}))$, $f \in F$, then let $r'_{i,j,k} : [f, i][d, 0] \to [f, 0][d, i](f, t_2)$, where t_2 is the type of C_2 – its rules will be given later. In this case f, $f \in F$, is also sent to C_2.

7. $r_{i,j,k} : c \leftrightarrow f$ with $c \in (O \setminus (F \cup \{e\}))$, $f \in F$, then let $r'_{i,j,k} : [c,i][f,0] \rightarrow [c,0][f,i](f',t_2)$, where t_2 is as above. In this case f', $f \in F$, is sent to C_2.
8. $r_{i,j,k} : f_1 \leftrightarrow f_2$ with $f_1, f_2 \in F$, then let $r'_{i,j,k} : [f_1,i][f_2,0] \rightarrow [f_1,0][f_2,i](f_1,t_2)(f'_2,t_2)$.
9. $r_{i,j,k} : f \leftrightarrow e$ with $f \in F$, then let $r'_{i,j,k} : [f,i] \rightarrow [f,0][e,i](f,t_2)$.
10. $r_{i,j,k} : e \leftrightarrow f$ with $f \in F$, then let $r'_{i,j,k} : [e,i][f,0] \rightarrow [f,i](f',t_2)$.

The set of rules of R_2 associated with type t_2 contains rules $p_f : ff' \rightarrow \lambda$, $f \in F$.

Each rule $r_{i,j,k}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, $1 \leq k \leq h$, interacts with two objects - either both from the cell B_i, when $r_{i,j,k}$ is an evolution rule, or one from B_i and the other from the environment, when it is a communication rule. Cases (1) and (2) above correspond to evolution rules and communication rules, respectively. For a rule $r_{i,j,k}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, $1 \leq k \leq h$, interacting with objects a, b from B_i, the corresponding rule $r'_{i,j,k}$, transforms $[a,i]$ into $[b,i]$. When the rule interacts with an object a from B_i and b from environment then the corresponding rule rewrites $[a,i]$ and $[b,0]$ into $[a,0],[b,i]$. Cases (3) to (10) present special cases of (2) when the objects are e or from F. When object e refers to an occurrence within the environment (3-5, 9) then the corresponding object in $r'_{i,j,k}$ does not appear, i.e., $[e,0]$ is not present in any of these rules. When an object f, $f \in F$, is sent to the environment (6, 8, 9) then the corresponding rules from R_1 have on the right hand side (f,t_2), which means that f is also sent to C_2, the output compartment. When f, $f \in F$, is brought into B_i from environment (7, 8 10) then the rules from R_1 have on the right hand side (f',t_2), $f \in F$; this means that f' is sent to C_2 and this will be used to erase f by using the rule r_f from R_2.

Since every program of Π consists of exactly h rules, we describe how the the rules of R_1 are constructed that correspond to the programs of Π with only evolution or communication rules.

We first need an auxiliary notation. For a rule of the form $r : u \rightarrow v$, where $u, v \in V^*$, where V is an alphabet, u and v are finite multisets over V, we define $lhs(r) = u$ and $rhs(r) = v$.

For each program $p_{i,j} =< r_{i,j,1} \ldots, r_{i,j,h} > \in P_i$, $1 \leq j \leq k_i$, $1 \leq i \leq n$, the set R_1 contains a rule of the form

$$p'_{i,j} : u_{i,j} \rightarrow v_{i,j},$$

where $u_{i,j} = lhs(r'_{i,j,1}) \ldots lhs(r'_{i,j,h})$ and $v_{i,j} = rhs(r'_{i,j,1}) \ldots rhs(r'_{i,j,h})$, and $r'_{i,j,k}$ $1 \leq i \leq n$, $1 \leq j \leq k_i$, $1 \leq k \leq h$, is associated to $r_{i,j,h}$ in accordance with one of the cases (1) – (10) above. (Note that $u_{i,j}$ and $v_{i,j}$ are strings representing finite multisets of objects).

The execution strategies of the types t_1, t_2 are given by $\sigma_1 = \{p'_{i,j} | 1 \leq i \leq n, 1 \leq j \leq k_i\}^\top$ and $\sigma_2 = \{p_f | f \in F\}^\top$. This means that each compartment executes the rules using maximal parallelism mode.

A computation in Π starts from the initial configuration $(x_E, x_1, \ldots, x_n)$ where $x_E = \lambda$ and $x_i = e^h$, $1 \leq i \leq n$. The initial configuration of $k\Pi$ is $([e^h,1] \ldots [e^h,n], \lambda)$, where $[e^h,i]$, $1 \leq i \leq n$, denotes h objects $[e,i]$. In

each cell B_i, $1 \leq i \leq n$, at most one program, consisting of h rules, can be applied, as B_i has always h objects from O. Similarly, in $k\Pi$ there are always h objects of the form $[a, i]$, where $a \in O$. For any configuration of Π, $(x_E, x_1, \ldots, x_n)$, where x_E is a multiset over $O \setminus \{e\}$ and x_i, $1 \leq i \leq n$, is a multiset over O with h objects. The corresponding configuration of $k\Pi$ is $([x_E, 0][x_1, 1] \ldots [x_n, n], y)$, where $[x_i, i]$, $1 \leq i \leq n$, denotes h objects $[a, i]$, for each a occurring in x_i (similarly for $[x_E, 0]$). A program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, is applied in B_i of Π if and only if $p'_{i,j}$ is applied in C_1 of $k\Pi$. Hence x_i, $1 \leq i \leq n$, appears in a configuration of Π if and only if $[x_i, i]$ appears in the corresponding configuration of $k\Pi$. The multiset x_E appears in the environment of Π if and only if $[x_E, 0]$ appears in $k\Pi$. The multiset y is given by $y = x_E uu'$, where $u = f_1 \ldots f_p$ and $u = f'_1 \ldots f'_p$. The multiset uu' is erased in the next step by applying rules of R_2 in a maximally parallel manner.

These show that the two mechanisms arrive at the same number of objects from F in the environment (P colony Π) and in C_2 (kP system $k\Pi$) in a halting computation, hence $N(\Pi) = N(k\Pi)$.

Now we are looking at P colonies with senders and consumers, aiming at simulating them by using again kP systems.

Theorem 2 *For every P colony $\Pi = (O, e, F, B_1, \ldots, B_n)$, $n \geq 1$, with senders and consumers, one can construct a kP system $k\Pi = (A, \mu, C_1, C_2, 2)$, such that $N(\Pi) = N(k\Pi)$ holds.*

Proof In this proof we follow similar ideas as in the proof of Theorem 1. Let us consider $\Pi = (O, e, F, B_1, \ldots, B_n)$, $n \geq 1$, a P colony with senders and consumers of size n and capacity 2. The result of a computation in Π is obtained in the environment, as the number of objects from F. The cell B_i, $1 \leq i \leq n$, has the initial multiset o_i consisting of two objects and each program $p_{i,j}$, $1 \leq j \leq k_i$, from the set of programs, P_i, is either a sender or a consumer. So, we have either consumer or producer cells.

We build, similar to the proof of Theorem 1, a kP system with two compartments

$$k\Pi = (A, \mu, C_1, C_2, 2)$$

where μ, C_1 and C_2 have the same meaning and A the same definition,

$$A = F \cup \{f' | f \in F\} \cup \{[a, i] | a \in O, 0 \leq i \leq n\},$$

as in the proof of Theorem 1.

Each compartment, C_i, $i = 1, 2$, has a type, t_i, and an initial multiset w_i. These multisets are $w_1 = [o_1, 1] \ldots [o_n, n]$ and $w_2 = \lambda$. Each o_i, $1 \leq i \leq n$, consists of two objects, $o_i = ab$. In this case $[o_i, i]$ means $[a, i][b, i]$.

In the sequel we define the set of rules R_1 and R_2, from compartments C_1 and C_2, respectively. For an arbitrary program $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, we will associate a rule $p'_{i,j} \in R_1$. As these programs are either senders or consumers, we will analyse each of these situations below. When

1. $p_{i,j} :< a_{out}; b \to cd >$ (sender) with $a, b, c, d \in O$, $a \notin F$, $a \neq e$, then R_1 has a rule $p'_{i,j} : [a, i][b, i] \to [a, 0][c, i][d, i]$.
2. $p_{i,j} :< f_{out}; b \to cd >$ (sender) with $f, b, c, d \in O$, $f \in F$, then R_1 has a rule $p'_{i,j} : [f, i][b, i] \to [f, 0][c, i][d, i](f, t_2)$, where t_2 is the type of C_2 – its rules will be given later. In this case f, $f \in F$, is also sent to C_2.
3. $p_{i,j} :< e_{out}; b \to cd >$ (sender) with $b, c, d \in O$, then R_1 has a rule $p'_{i,j} : [e, i][b, i] \to [c, i][d, i]$.
4. $p_{i,j} :< a_{in}; bc \to d >$ (consumer) with $a, b, c, d \in O$, $a \notin F$, $a \neq e$, then R_1 has a rule $p'_{i,j} : [a, 0][b, i][c, i] \to [a, i][d, i]$.
5. $p_{i,j} :< f_{in}; bc \to d >$ (consumer) with $b, c, d \in O$, $f \in F$, then R_1 has a rule $p'_{i,j} : [f, 0][b, i][c, i] \to [f, i][d, i](f', t_2)$. In this case f', $f \in F$, is sent to C_2.
6. $p_{i,j} :< e_{in}; bc \to d >$ (consumer) with $b, c, d \in O$, then R_1 has a rule $p'_{i,j} : [b, i][c, i] \to [e, i][d, i]$.

The set of rules R_2 associated with type t_2 contains rules $p_f : ff' \to \lambda$, $f \in F$.

Each program $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, includes a rule interacting only with objects, b, c, d, from the cell B_i. The corresponding rule $p'_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, cases (1-6), includes $[b, i], [c, i], [d, i]$, respectively. The other rule that appears in each program $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, either sends an a to the environment (1) or consumes it from the environment (4). In these cases the corresponding rule $p'_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, transforms $[a, i]$ into $[a, 0]$ (1) or the other way around (4). When $a = f$, $f \in F$, then the rule $p'_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, corresponding to the sender (2) includes on the right hand side (f, t_2), indicating that f is also sent to C_2, the output compartment. When f, $f \in F$, is brought into B_i from environment, the consumer, then the corresponding rule $p'_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, (5) has on the right hand side (f', t_2), $f \in F$; this means that f' is sent to C_2 and this will be used to erase f by using the rule p_f from R_2. When object e refers to an occurrence within the environment (3, 6) then the corresponding object in $p'_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, does not appear, i.e., $[e, 0]$ is not present in any of these rules.

The execution strategies of the types t_1, t_2, as in the proof of Theorem 1, are given by $\sigma_1 = \{p'_{i,j} | 1 \leq i \leq n, 1 \leq j \leq k_i\}^\top$ and $\sigma_2 = \{p_f | f \in F\}^\top$, i.e., in each compartment the rules are executed using maximal parallelism mode.

A computation in Π starts from the initial configuration $(x_E, x_1, \ldots, x_n)$ where $x_E = \lambda$ and $x_i = ee$, $1 \leq i \leq n$. The initial configuration of $k\Pi$ is $([e, 1][e, 1] \ldots [e, n][e, n], \lambda)$. In each cell B_i, $1 \leq i \leq n$, at most one program, consisting of either a sender or a consumer, can be applied, as B_i has always two objects a, b. Similarly, in $k\Pi$ there are always two object $[a, i], [b, i]$. For any configuration of Π, $(x_E, x_1, \ldots, x_n)$, where x_E is a multiset over $O \setminus \{e\}$ and x_i, $1 \leq i \leq n$, is a multiset over O with two objects. The corresponding configuration of $k\Pi$ is $([x_1, 1] \ldots [x_n, n], y)$. A program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, is applied in B_i of Π if and only if $p'_{i,j}$ is applied in C_1 of $k\Pi$. Hence x_i, $1 \leq i \leq n$, appears in a configuration of Π if and only if $[x_i, i]$ appears in the corresponding configuration of $k\Pi$. The multiset y is given by x_E and maybe

a multiset uu', where $u = f_1 \dots f_p$ and $u = f'_1 \dots f'_p$. The multiset uu' is erased in the next step by applying rules of R_2 in a maximal parallel manner.

These show that the two mechanisms arrive at the same number of objects from F in the environment (P colony Π) and in C_2 (kP system $k\Pi$) in a halting computation, hence $N(\Pi) = N(k\Pi)$.

Next we show that kernel P systems simulate P colonies with evolving environment as well. Notice that in the case of these P colony variants the possibly dynamically changing environment is expected to alter the computational power of the P colony.

Theorem 3 *For every P colony* $\Pi = (O, e, F, o_E, D_E, B_1, \dots, B_n)$, $n \geq 1$, *with evolving environment and with capacity* 2 *one can construct a kP system* $k\Pi = (A, \mu, C_1, C_2, 2)$, *such that* $N(\Pi) = N(k\Pi)$ *holds.*

Proof Let us consider $\Pi = (O, e, F, o_E, D_E, B_1, \dots, B_n)$, $n \geq 1$, a P colony of size n and capacity 2. Without the loss of generality, we may assume that cells B_i $1 \leq m \leq n$ are of type sender and consumer (m is an even number and Π has as many sender cells as consumer cells) and cells B_j, $m+1 \leq j \leq n$ have only evolution and/or communication rules.

The cell B_i, $1 \leq i \leq n$, has the initial multiset, o_i, consisting of two objects and each program, $p_{i,j}$, $1 \leq j \leq k_i$, in the set of programs, P_i, depending of the type of B_i consists of one or two rules.

To prove the statement, we construct a kP system with two compartments

$$k\Pi = (A, \mu, C_1, C_2, 2)$$

where C_1 simulates the computations of Π and C_2 collects the result of $k\Pi$ that is equal to the result of Π.

The objects of the kP system are $A = F \cup \{f' | f \in F\} \cup \{[a, i] | a \in O, 0 \leq i \leq n\}$, as in the case of the proofs of Theorem 1 and Theorem 2. Each $a \in O$, when it appears in cell B_i, $1 \leq i \leq n$, or in the the environment, 0, it will be denoted by $[a, i]$, $1 \leq i \leq n$, or $[a, 0]$, respectively.

The membrane structure, μ, provides a connection between the two compartments, C_1 and C_2, allowing them to exchange objects, where $C_i = (t_i, w_i)$, $i = 1, 2$. Each t_i, $1 \leq i \leq 2$, the type of compartment C_i, is given by $t_i = (R_i, \sigma_i)$, where R_i is the set of rules and σ_i the execution strategy that are defined analogously to the previous proofs, with slight changes for σ_1 as it will be shown when this execution strategy is defined. The second component of C_i, the initial multiset of the compartment, is given as $w_1 = [o_1, 1] \dots [o_n, n]$, where each o_i, $1 \leq i \leq n$, consists of two objects a, b and $[o_i, i]$ denotes $[a, i][b, i]$, and $w_2 = \lambda$. We will look at these initial multisets when the behaviour of $k\Pi$ will be discussed.

As in the previous proofs the compartment C_1 will host the objects and rules corresponding to those of Π and C_2 will collect the results of the computation. In this case the simulation of the P colony Π in C_1 will be such that in each of the computation step in Π are first applied the programs in

cells B_i, $1 \leq i \leq n$, and if there is at least one cell with at least an applicable program then the $0L$ scheme D_E is also used. This two stage application of the programs followed by the use of the $0L$ scheme D_E will be simulated in C_1 by a special definition of the execution strategy σ_1.

We define the rule sets R_1, R_2 of kH in several steps.

We start by defining rule set R'_1, a subset of R_1, which is associated with the type t_1 of C_1 that corresponds to the programs of cells B_i, $1 \leq i \leq m$, where $m \leq n$. The reader may easily notice that these rules can be obtained exactly in the same manner as the rules were contructed in the proof of Theorem 2, in case of P colonies with senders and consumers. Thus, we define for an arbitrary program $p_{i,j} \in P_i$, $1 \leq i \leq m$, $1 \leq j \leq k_i$, a rule $p'_{i,j} \in R'_1$ as follows:

1. $p_{i,j} :< a_{out}; b \to cd >$ with $a, b, c, d \in O$, $a \notin F$, $a \neq e$, then R'_1 has a rule $p'_{i,j} : [a, i][b, i] \to [a, 0][c, i][d, i]$.
2. ' $p_{i,j} :< f_{out}; b \to cd >$ with $f, b, c, d \in O$, $f \in F$, then R'_1 has a rule $p'_{i,j} : [f, i][b, i] \to [f, 0][c, i][d, i](f, t_2)$, where t_2 is the type of C_2 – as in the previous prrofs, its rules will be given later. In this case f, $f \in F$, is also sent to C_2.
3. $p_{i,j} :< e_{out}; b \to cd >$ with $b, c, d \in O$, then R'_1 has a rule $p'_{i,j} : [e, i][b, i] \to [c, i][d, i]$.
4. $p_{i,j} :< a_{in}; bc \to d >$ with $a, b, c, d \in O$, $a \notin F$, $a \neq e$, then R'_1 has a rule $p'_{i,j} : [a, 0][b, i][c, i] \to [a, i][d, i]$.
5. $p_{i,j} :< f_{in}; bc \to d >$ with $b, c, d \in O$, $f \in F$, then R'_1 has a rule $p'_{i,j} : [f, 0][b, i][c, i] \to [f, i][d, i](f', t_2)$. In this case f', $f \in F$, is sent to C_2.
6. $p_{i,j} :< e_{in}; bc \to d >$ with $b, c, d \in O$, then R'_1 has a rule $p'_{i,j} : [b, i][c, i] \to [e, i][d, i]$.

The correspondence of the above rules of kH and the programs of cells B_i, $1 \leq i \leq m$, and their role in the computation can easily be obtained by the same reasoning as we used in the proof of Theorem 2.

Next we define the rule set R''_1, a subset of R_1, which is associated with the type t_1 of C_1 that corresponds to the programs of cells B_i, $m+1 \leq i \leq n$. Notice first that any program $p_{i,j} \in P_i$, $m+1 \leq i \leq n$, $1 \leq j \leq k_i$, consists of two rules $r_{i,j,1}$ and $r_{i,j,2}$, where every rule can be an evolution rule or a communication rule. Furthermore, if a program is applied, then both rules should be performed in parallel.

Analogously to the proof of Theorem 1, first we will define rules $r'_{i,j,k}$ corresponding to rules $r_{i,j,k}$ from programs $p_{i,j} \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, $k = 1, 2$. When

1. $r_{i,j,k} : a \to b$ with $a, b \in O$, then let $r'_{i,j,k} : [a, i] \to [b, i]$.
2. $r_{i,j,k} : c \leftrightarrow d$ with $c, d \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [c, i][d, 0] \to [c, 0][d, i]$.
3. $r_{i,j,k} : c \leftrightarrow e$ with $c \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [c, i] \to [c, 0][e, i]$.
4. $r_{i,j,k} : e \leftrightarrow d$ with $d \in (O \setminus (F \cup \{e\}))$, then let $r'_{i,j,k} : [e, i][d, 0] \to [d, i]$.
5. $r_{i,j,k} : e \leftrightarrow e$ then let $r'_{i,j,k} : [e, i] \to [e, i]$.

6. $r_{i,j,k} : f \leftrightarrow d$ with $d \in (O \setminus (F \cup \{e\}))$, $f \in F$, then let $r'_{i,j,k} : [f,i][d,0] \rightarrow [f,0][d,i](f,t_2)$, where t_2 is the type of C_2 – its rules will be given later. In this case f, $f \in F$, is also sent to C_2.
7. $r_{i,j,k} : c \leftrightarrow f$ with $c \in (O \setminus (F \cup \{e\}))$, $f \in F$, then let $r'_{i,j,k} : [c,i][f,0] \rightarrow [c,0][f,i](f',t_2)$, where t_2 is as above. In this case f', $f \in F$, is sent to C_2.
8. $r_{i,j,k} : f_1 \leftrightarrow f_2$ with $f_1, f_2 \in F$, then let $r'_{i,j,k} : [f_1,i][f_2,0] \rightarrow [f_1,0][f_2,i](f_1,t_2)(f'_2,t_2)$.
9. $r_{i,j,k} : f \leftrightarrow e$ with $f \in F$, then let $r'_{i,j,k} : [f,i] \rightarrow [f,0][e,i](f,t_2)$.
10. $r_{i,j,k} : e \leftrightarrow f$ with $f \in F$, then let $r'_{i,j,k} : [e,i][f,0] \rightarrow [f,i](f',t_2)$.

The meaning of the rules above is the same that of the corresponding rules in the proof of Theorem 1, therefore we omit the explanations. Since every program of Π consists of exactly two rules, the rules of $k\Pi$ that correspond to the programs of Π with evolution and/or communication rules are constructed analogously to the way that the set of rules R_1 defined in the proof of Theorem 1, taking capacity $h = 2$.

Now we define the rules that appear in R''_1.

For each program $p_{i,j} =< r_{i,j,1}, r_{i,j,2} >$, $m+1 \leq i \leq n$, $1 \leq j \leq k_i$, from P_i of the cell B_i, the rules $r'_{i,j,1}$ and $r'_{i,j,2}$ are built as shown above. Denoting $u_{i,j} = lhs(r'_{i,j,1})lhs(r'_{i,j,2})$ and $v_{i,j} = rhs(r'_{i,j,1})rhs(r'_{i,j,2})$, as in the proof of Theorem 1, we add $p'_{i,j} : u_{i,j} \rightarrow v_{i,j}$ to R''_1.

Next we define the rule set R'''_1, a subset of R_1, which simulates the rules in rule set P_E of the multiset $0L$ scheme D_E. For each rule of the form $r : a \rightarrow u \in P_E$, where $a \in O$ and $u = b_1 \dots b_s$, $b_l \in O$, $1 \leq l \leq s$, or $p : a \rightarrow \lambda \in P_E$ we assign a rule of the form $r' : [a,0] \rightarrow [b_1,0] \dots [b_s,0]\{g\}$ or $p' : [a,0] \rightarrow \lambda\{g\}$, respectively. The guard, defined below, will impose constraints on the application of the rules from R'''_1, such that these are applied if and only if the rules associated to a program from at least one cell B_i, $1 \leq i \leq n$, are applicable to the current multiset. With this constraint we make sure that the rules of R'''_1 will stop being applied when no rule from $R'_1 \cup R''_1$ is applicable, and consequently the computation stops.

The rule set R_1 is given by $R'_1 \cup R''_1 \cup R'''_1$.

In order to define the guard g that appears in every rule from R'''_1 we introduce a notation. For a rule $r \in R'_1 \cup R''_1$, $lhs(r)$ is a multiset, that appears on the left hand side of the rule. If this is a multiset over $\{a_1, \dots, a_t\}$, then it might be written as $a_1^{p_1} \dots a_t^{p_t}$, with $p_i \geq 1$, $1 \leq i \leq t$. We introduce the guard $g_{lhs(r)}$ denoting $\geq a_1^{p_1} \wedge \dots \wedge \geq a_t^{p_t}$. This guard is true for a multiset w if and only if w contains at least p_i occurrences of a_i, $1 \leq i \leq t$. Having $g_{lhs(r)}$ defined for every $r \in R'_1 \cup R''_1$, one can define the guard g as follows

$$g = g_{lhs(p'_{1,1})} \vee \dots \vee g_{lhs(p'_{1,k_1})} \vee \dots \vee g_{lhs(p'_{n,1})} \vee \dots \vee g_{lhs(p'_{n,k_n})}.$$

The execution strategy of type t_1 is $\sigma_1 = \{r | r \in R'_1 \cup R''_1\}^{\top}\{r | r \in R'''_1\}^{\top}$. This means that first are applied in maximal parallel manner the rules from $R'_1 \cup R''_1$ followed by rules from R'''_1 applied also in maximal parallel way.

The set of rules of R_2 associated with type t_2 contains rules $p_f : ff' \rightarrow \lambda$, $f \in F$.

The execution strategy of the type t_2 is given by $\sigma_2 = \{p_f | f \in F\}^\top$, i.e., the rules use the maximally parallel application mode.

A computation in Π starts from the initial configuration $(o_E, o_1, \ldots, o_n)$ where $o_E \in (O \setminus \{e\})^*$ and $o_i = ee$, $1 \leq i \leq n$. The initial configuration of $k\Pi$ is $([o_E, 0][e, 1][e, 1] \ldots [e, n][e, n], \lambda)$. In each cell B_i, $1 \leq i \leq n$, at most one program, consisting of one or two rules, can be applied, and B_i has always two objects inside. Those objects which are in the environment but are not affected by the activity of the cells are changed according to the rules in P_E. Similarly, in $k\Pi$ for any i, $1 \leq i \leq n$, there are always two objects of the form $[a, i]$. Any configuration of Π has the form $(x_E, x_1, \ldots, x_n)$, where x_E is a multiset over $O \setminus \{e\}$ and x_i, $1 \leq i \leq n$, is a multiset over O with two objects. The corresponding configuration of $k\Pi$ is $([x_E, 0][x_1, 1] \ldots [x_n, n], y)$, where $[x_i, i]$, $1 \leq i \leq n$, denotes two objects $[a, i]$, as defined earlier in the proof. The object $[x_E, 0]$, as in the proof of Theorem 1, corresponds to the x_E from the environment of Π. It is easy to see that a program $p_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, is applied in B_i of Π if and only if the corresponding rule $p'_{i,j}$ is applied in C_1 of $k\Pi$. Hence x_i, $1 \leq i \leq n$, appears in a configuration of Π if and only if $[x_i, i]$ appears in the corresponding configuration of $k\Pi$ and x_E is in the environment of Π if and only if $[x_E, 0]$ appears in $k\Pi$. The multiset y is given by $y = x_E u u'$, where $u = f_1 \ldots f_p$ and $u' = f'_1 \ldots f'_p$. If no more rule simulating a program of Π or corresponding to a rule of D_E is applicable, then the multiset uu' is erased in the next step by applying rules of R_2 in a maximally parallel manner.

Thus, the two mechanisms have the same number of objects from F in the environment (P colony Π) and in C_2 (kP system $k\Pi$) in a halting computation, hence $N(\Pi) = N(k\Pi)$.

4 Producer/consumer example

The producer/consumer paradigm consists of a system with two processors, a producer and a consumer, synchronising through a buffer of length 1 (accepting no more than an element). This has been modelled in different formalisms, including Petri nets [27] and generalised membrane systems using rewriting rules [1]. Recently, another approach based on generalised communicating P systems has been presented [17]. A slightly different version of this problem, using numerical P systems has been considered [24, 25].

Here we model this problem using P colonies with senders and consumers and then consider it with kP systems. In this approach we use P colonies with senders and consumers having cells with both sender and consumer programs as they provide much simpler solutions to our problem.

The producer/consumer problem considered in this section can be described as follows: when the producer is informed that the buffer is empty, a signal that the buffer becomes full, followed by the current symbol are sent out to the consumer. After the symbol is consumed, the consumer responds back to the producer that the buffer is now empty.

We consider a P colony with senders and consumers, $\Pi = (O, e, F, B_1, B_2)$, where the alphabet is $O = \{a, b, s, v, f\}$ and B_1 the producer cell and B_2 the consumer. Intially, $o_1 = vf$ and $o_2 = ab$. Please note that this is slightly different from the case studied before, whereby the initial multisets contain only multisets over $\{e\}$. It is easy to be observed that one can get the values for o_1 and o_2 with some simple initialisation rules starting from ee in both cells. We leave this as an exercise for the reader.

The set of final objects is not significant for this problem. The P colony with senders and consumers works as follows: the sender after receiving the signal that the buffer is empty (symbol v - void, is received), sends to the environment a signal f - full, that the buffer will become full and then the current symbol, s. These are sent to the environment and then retrieved by the consumer. We provide now the set of programs for the two cells.

- P_1, producer
  - $p_1 :< f_{out}; v \rightarrow sf >$;
  - $p_2 :< s_{out}; v \rightarrow af >$;
  - $p_3 :< v_{in}; af \rightarrow f >$;
- P_2, consumer
  - $c_1 :< f_{in}; ab \rightarrow a >$;
  - $c_2 :< s_{in}; af \rightarrow v >$;
  - $c_3 :< v_{out}; s \rightarrow ab >$.

If (x_E, x_1, x_2) and (y_E, y_1, y_2) are two configurations and p and p' the programs used to pass from the first to the second configuration then we write $(x_E, x_1, x_2) \Rightarrow^{p,p'} (y_E, y_1, y_2)$. When one of the programs is not applicable then it is omitted.

One can write the following sequence of configurations starting from the initial one:

$$z_1 = (\lambda, vf, ab) \Rightarrow^{p_1} z_2 = (f, sf, ab) \Rightarrow^{p_2,c_1} z_3 = (s, af, af) \Rightarrow^{c_2}$$

$$z_4 = (\lambda, af, sv) \Rightarrow^{c_3} z_5 = (v, af, ab) \Rightarrow^{p_3} z_6 = (\lambda, vf, ab).$$

This shows that when the buffer is empty (v in B_1 in configuration z_1) then an f is sent to the environment (f in configuration z_2) and then the symbol s is sent to the environment (configuration z_3). These are then consumed by B_2. Symbol v - corresponding to void buffer, is sent to the environmnet by B_2 (configuration z_5). In configuration z_6, the same as z_1, the void buffer is signalled to the producer, B_1.

Using the construction from the proof of Theorem 2 one can get a kP system that models the producer/consumer problem. This is left to the reader as a simple exercise. We prefer to provide a kP system with two compartments and equivalent to the one obtained from the P colony. The kP system is given by $k\Pi = (A, \mu, C_1, C_2)$. The output compartment is not significant and hence not considered above. The alphabet of objects is $A = \{a, b, c, v, f, s\}$, μ describes the connection between C_1 and C_2. The compartments are $C_1 = (p, w_1)$, corresponding to producer, and $C_2 = (c, w_2)$, corresponding to consumer, where

$w_1 = w_2 = a$. The types p and c include the sets of rules R_1 and R_2, respectively. These are described below.

- R_1, producer
  - $p'_1 : a \to b(f, c)$;
  - $p'_2 : b \to c(s, c)$;
  - $p'_3 : vc \to a$;
- R_2, consumer
  - $c'_1 : fa \to b$;
  - $c'_2 : sb \to c$;
  - $c'_3 : c \to a(v, p)$.

The rules are applied with maximal parallelism in each compartment.

We use a notation similar to the one used for P colonies to define a sequence of configurations. We use two multisets of rules instead of programs. The following sequence can be obtained starting from the initial configuration:

$$z'_1 = (a, a) \Rightarrow^{\{p'_1\}} z'_2 = (b, fa) \Rightarrow^{\{p'_2\},\{c'_1\}} z'_3 = (c, sb) \Rightarrow^{\{c'_2\}}$$

$$z'_4 = (c, c) \Rightarrow^{\{c'_3\}} z'_5 = (vc, a) \Rightarrow^{\{p'_3\}} z'_6 = (a, a).$$

Although in the case of kP systems there is no environment, the components communicating directly, one can observe a very similar distribution of the key symbols - v and f related to the buffer status and s, generated by producer and received by consumer.

For the kP system model we are able to validate it by using the tools available within the kPWorkbench - a specification language for such models and model checkers [7]. We can simulate and also formally verify certain properties of the model. We present here some of the properties that we have considered for this model.

One can observe that the consumer (compartment C_2) consumes the current symbol, s, only after it is notified that the buffer is full, i.e., f appears in C_2. This can be verified by considering the following property
$C_2.f > 0$ `followed-by` $C_2.s > 0$
and this can be expressed by the following specification of the model checker
`AG (C2.f>0 -> EF C2.s>0)`,
which is true. The fact that the system has a cyclic behaviour, returning back to its initial state, containing a in each of the two compartments, can be also formally proved by using
`infinitely-often` $C_1.a > 0$.

This is expressed as
`AG(EF C1.a>0)`,
which is also true. A similar property can be provided for $C_2.a$.

With such properties one can prove the correctness of our model specification, making sure the model describes accurately the problem.

5 Conclusions

In this paper we have investigated the relationships among three classes of P colonies and kernel P systems. We have also showed how a synchronisation problem is represented by these formalisms. Some future research directions will include the possibility of enriching P colony models with features of the kernel P systems that might enhance the power and efficiency of the former. Another research avenue might be that of investigating the relationships with other P system models that will prove the flexibility of these classes of models in describing the behaviour of other computational models.

Acknowledgements The work of EC-V was supported by grant No. K 120558 of the NKFIH- National Research, Development, and Innovation Office, Hungary, MG and RL acknowledge the support provided by the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number PN-III-P4- ID-PCE-2016-0210).

References

1. Bernardini, F., Gheorghe, M., Margenstern, M., Verlan, S.: Producer/consumer in membrane systems and Petri nets. In: 3rd Conference on Computability in Europe, CiE 2007, *Lecture Notes in Computer Science*, vol. 4495, pp. 43 – 52. Springer (2017)
2. Ciencialová, L., Cienciala, L., Sosík, P.: P colonies with evolving environment. In: Leporati et al. [19], pp. 151–164
3. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., Sosík, P.: P colonies. Bulletin of the of the International Membrane Computing Society **2**, 129–156 (2016)
4. Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., Vaszil, G.: Variants of P colonies with very simple cell structure. International Journal of Computers, Communications and Control **4**(3), 224 – 233 (2009)
5. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A.: Computing with cells in environment: P colonies. Multiple-Valued Logic and Soft Computing **12**(3-4), 201–215 (2006)
6. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierla, L.: Model checking kernel P systems. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Y. Rogozhin, G. Rozenberg, A. Salomaa (eds.) Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 8340, pp. 151–172. Springer (2013)
7. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S.: Kernel P systems modelling, testing and verification - sorting case study. In: Leporati et al. [19], pp. 233–250
8. Gheorghe, M., Ipate, F., Dragomir, C.: Kernel P systems. Tenth Brainstorming Week on Membrane Computing pp. 153–170 (2012)
9. Gheorghe, M., Ipate, F., Dragomir, C., Mierla, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P systems - version I. Eleventh Brainstorming Week on Membrane Computing (11BWMC) pp. 97–124 (2013)
10. Gheorghe, M., Ipate, F., Konur, S.: Kernel P systems and relationships with other classes of P systems. In: M. Gheorghe, I. Petre, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.) Multidisciplinary Creativity, pp. 64–76. Spandugino Publishing House (2015)
11. Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M.J., Turcanu, A., Valencia-Cabrera, L., García-Quismondo, M., Mierla, L.: 3-col problem modelling using simple kernel P systems. International Journal of Computer Mathematics **90**(4), 816–830 (2013)
12. Gheorghe, M., Konur, S., Ipate, F.: Kernel P systems and stochastic P systems for modelling and formal verification of genetic logic gates. In: A. Adamatzky (ed.) Advances in Unconventional Computing: Volume 1: Theory, pp. 661–675. Springer International Publishing (2017)

13. Gheorghe, M., Konur, S., Ipate, F., Mierla, L., Bakir, M.E., Stannett, M.: An integrated model checking toolset for kernel P systems. In: G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (eds.) Membrane Computing - 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 9504, pp. 153–170. Springer (2015)
14. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX), pp. 82–86. Boston, Massachusetts, USA (2004)
15. Kelemenová, A.: P colonies. In: Păun, Gh., G. Rozenberg, A. Salomaa (eds.) The Oxford Handbook of Membrane Computing, pp. 584–594. OUP Oxford (2010)
16. Krishna, S.N., Gheorghe, M., Dragomir, C.: Some classes of generalised communicating P systems and simple kernel P systems. In: P. Bonizzoni, V. Brattka, B. Löwe (eds.) The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7921, pp. 284–293. Springer (2013)
17. Krishna, S.N., Gheorghe, M., Ipate, F., Csuhaj-Varjú, E., Ceterchi, R.: Further results on generalised communicating P systems. Theoretical Computer Science **701**, 146 – 160 (2017)
18. Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. Fundamenta Informaticae **131**(3 – 4), 451 – 464 (2014)
19. Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.): Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 10105. Springer (2017)
20. Mutyam, M., Krithivasan, K.: Generalized normal form for rewriting P systems. Acta Informatica **38**(10), 721 – 734 (2002)
21. Mutyam, M., Prakash, V.J., Krithivasan, K.: Rewriting tissue P systems. Journal of Universal Computer Science **10**(9), 1250 – 1271 (2004)
22. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)
23. Păun, Gh.: Membrane Computing - An Introduction. Springer (2002)
24. Păun, Gh., Păun, R.: Membrane computing as a framework for modeling economic processes. In: SYNASC05, Romania, IEEE Press, pp. 11 – 18 (2005)
25. Păun, Gh., Păun, R.: Membrane computing and economics. In: Păun, Gh., G. Rozenberg, A. Salomaa (eds.) The Oxford Handbook of Membrane Computing, pp. 632–644. OUP Oxford (2010)
26. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
27. Reisig, W.: Elements of distributed algorithms: modeling and analysis with Petri nets. Springer (1998)